Lab 11

Comp 11 - Summer Session — My Stack

11.1 Description

In this lab we will implement the stack data structure. Much of the code has been implemented for you–but you need to practice typing in the code and building the class first and foremost. This time, you will also build your program in multiple files(separating the interface from the implementation).

Our objectives are the following:

- Compile multiple files together
- Implement the push member function of stack
- Become even more comfortable with templates in classes!

11.2 Files

You may use the following code to help get you started. You will notice that this class is also templated for you, so you can use CompVector on whatever data type you like!

```cpp
// Compile this program as follows:
// clang++ main.cpp stack.cpp -o lab11

#include "CompStack.h" // our stack
#include <stack>  // C++ standard implementation

// Include other headesr
#include <iostream>

int main(){

    CompStack<float> ourStack;
    std::stack<float> cppStack;

    // Add items to our stack
    for(float f =0; f < 1; f = f + 0.1){
        ourStack.push(f);
        cppStack.push(f);
    }

    // Test that all items are added and accounted for
    std::cout << "Our stack size: " << ourStack.size() << "\n";
    std::cout << "cpp stack size: " << cppStack.size() << "\n";

    // Pop off some items and print them
    // Ensure that they match
    while(!ourStack.empty()){
        std::cout << ourStack.top() << " = " << cppStack.top() << "
    \n";
        ourStack.pop();
        cppStack.pop();
    }

    // Make sure pushing pushes on the top
    ourStack.push(99.99);
    cppStack.push(99.99);
    // Check top item
    std::cout << "Our stack top: " << ourStack.top() << "\n";
    std::cout << "cpp stack top: " << cppStack.top() << "\n";

    // Push even more things!
    for(float f =10; f < 11; f = f + 0.1){
        ourStack.push(f);
        cppStack.push(f);
    }
    // Check top item
    std::cout << "Our stack top: " << ourStack.top() << "\n";
    std::cout << "cpp stack top: " << cppStack.top() << "\n";

    // Final test that all items are added and accounted for
    std::cout << "Our stack size: " << ourStack.size() << "\n";
    std::cout << "cpp stack size: " << cppStack.size() << "\n";
    return 0;
}
```

Listing 11.1: Stack main.cpp

```
#ifndef COMPSTACK_H
#define COMPSTACK_H

template <class T>
class CompStack{
// Private variables not visible to user
    int pos;        // Where we are in our internal array
                    // For a stack, this also is the size
    int capacity;   // How many items we can store

    T* elements;    // Array of data we are storing

public: // Public functions available to user
    CompStack();
    ~CompStack();

    // Check if the stack is empty
    bool empty();
    // Get the stack size
    int size();
    // Return the top of the stack
    T top();
    // Add an item to the top of the stack
    void push(T element);
    // Remove an item
    void pop();
};

#endif
```

Listing 11.2: Stack CompStack.h

```cpp
#include "CompStack.h" // Include our interface
#include <iostream>    // For debugging if we need

// Constructor
template <class T>
CompStack<T>::CompStack(){
    // Set our initializers to zero.
    elements = NULL;
    capacity = 0;
    pos = 0;
}

// Destructor
template <class T>
CompStack<T>::~CompStack(){
    delete[] elements;
}

// Return the stack size
template <class T>
int CompStack<T>::size(){
     return pos;
}

template <class T>
bool CompStack<T>::empty(){
    if(0==pos){
        return true;
    }
    return false;
}

// Return the top element
template <class T>
T CompStack<T>::top(){
    // If our stack is not empty
    // Just return the top item
    return elements[pos-1];
}

// Add an element to the top of the stack
template <class T>
void CompStack<T>::push(T element){

// Hmm, what goes here?


}

// Remove the last item in the stack
// Actually no removal here--ask yourself why?
template <class T>
void CompStack<T>::pop(){
    // Decrement pos
    if(pos > 0){
        pos--;
    }
```

```
58 }
59
60
61 // Some compiler magic here, with templates,
62 // we can force C++ to tell us what types to generate
63 // This must be done when we split the interface from the
       implementation.
64 // If we only implement in the header, then this is not necesseary.
65 template class CompStack<float>;
66 template class CompStack<int>;
67 template class CompStack<double>;
68 template class CompStack<bool>;
```

Listing 11.3: Stack CompStack.cpp

11.3 Output

```
mike:lab11$ ./lab11
Our stack size: 10
cpp stack size: 10
0.9 = 0.9
0.8 = 0.8
0.7 = 0.7
0.6 = 0.6
0.5 = 0.5
0.4 = 0.4
0.3 = 0.3
0.2 = 0.2
0.1 = 0.1
0 = 0
Our stack top: 99.99
cpp stack top: 99.99
Our stack top: 10.9
cpp stack top: 10.9
Our stack size: 11
cpp stack size: 11
```

11.4 Refresher

You can use std::stack to understand the behavior of the data structor.

11.5 Submission

```
provide comp11 lab11 main.cpp CompStack.cpp CompStack.h README
```

Listing 11.4: Submit Assignment

11.6 Going Further

Did you enjoy this lab? Want to try out some additional commands to go further?

- Try implementing the pop method such that it reduces capacity at some threshold. You can add additional member variables if you like to do this.
- How would you handle errors in this lab?